Improved Superblock Optimization in GCC

Robert Kidd

University of Illinois

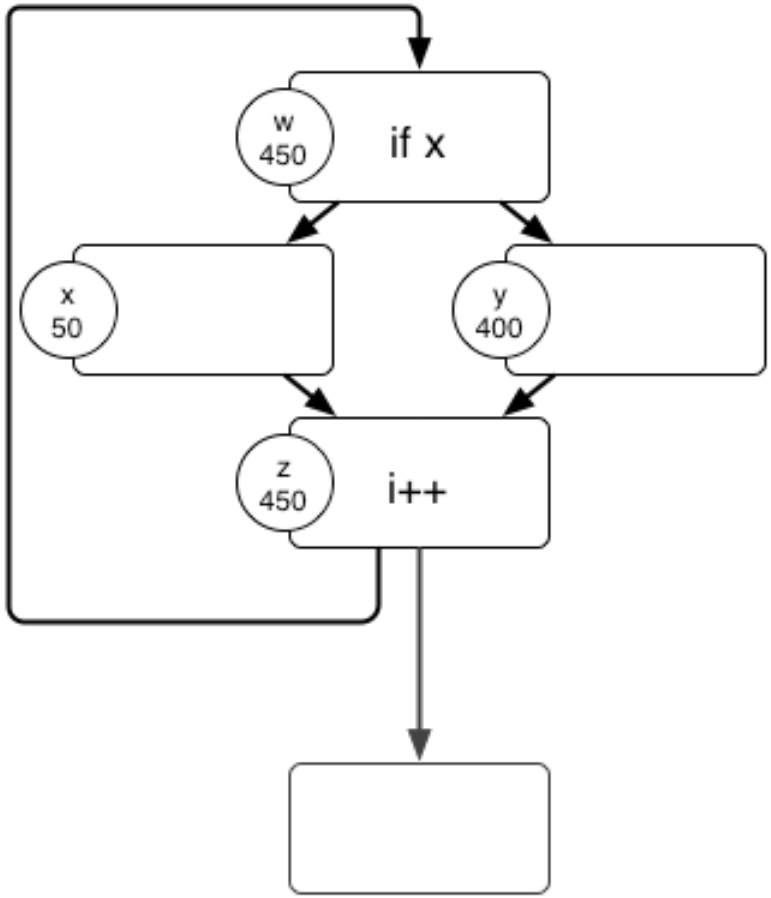
4/17/07

IMPACT
ADVANCED COMPILER TECHNOLOGY

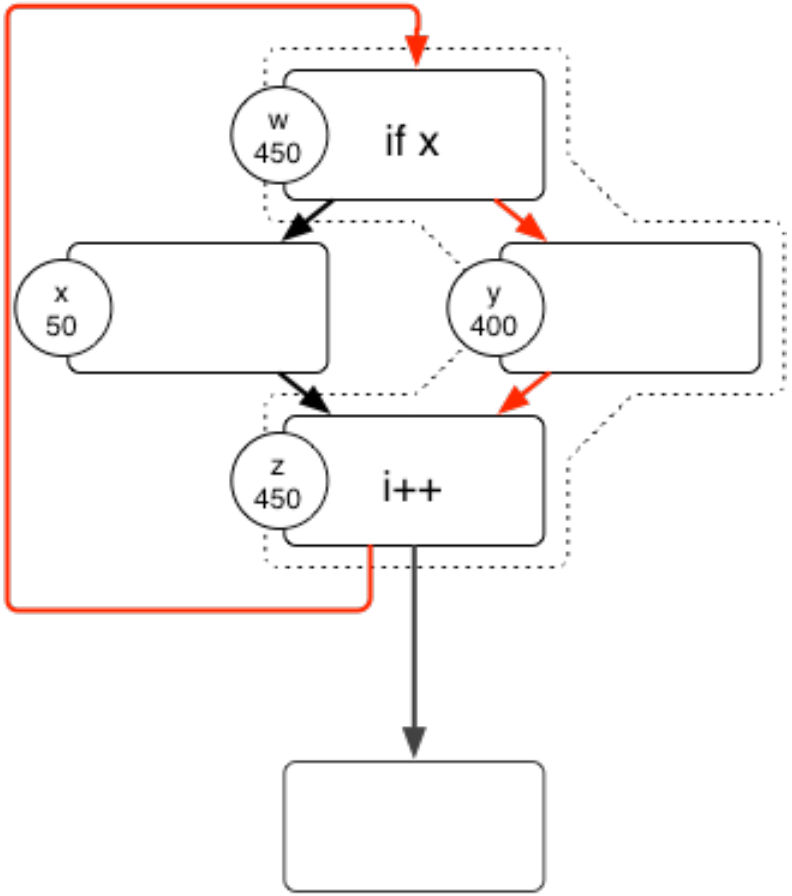
Introduction

- Project aims to replicate OpenIMPACT's structural compilation model in GCC
- Structural compilation: control flow specialization followed by traditional optimization
- Moved Superblock formation to the Tree-SSA level, implementing Branch Target Expansion

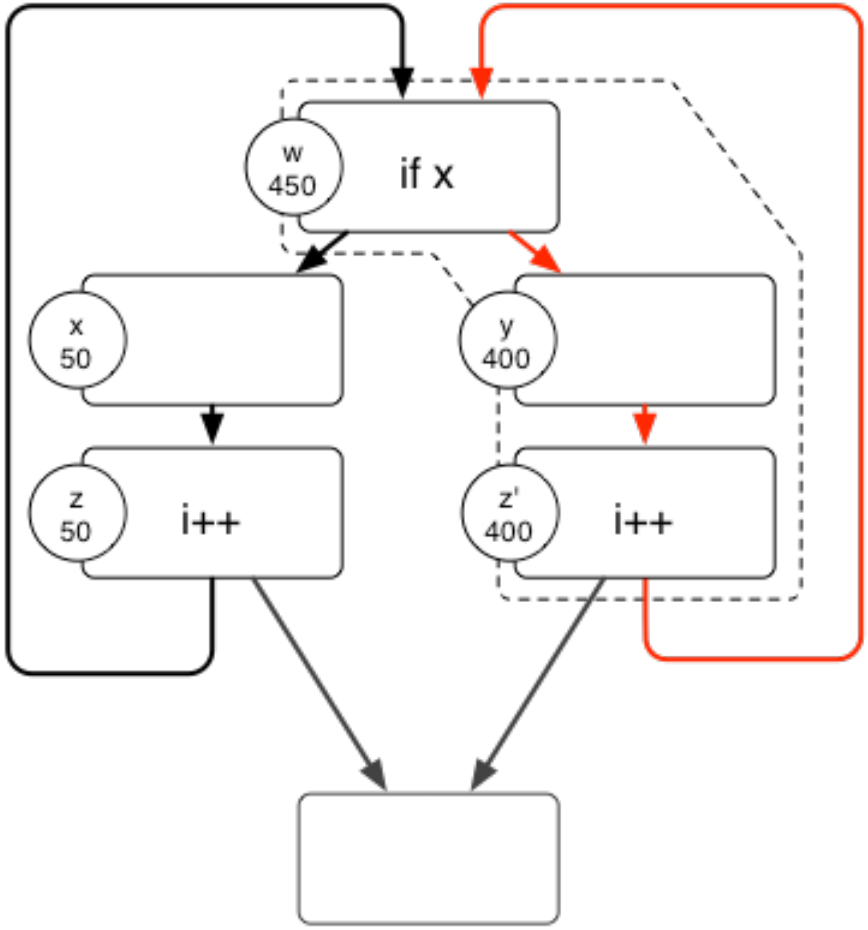
Introduction to Superblock Formation



Trace Formation



Tail Duplication



Superblock optimization

- Eliminating side entrances through tail duplication allows most of the benefit of trace scheduling with less bookkeeping
  - Effectively generates compensation code before moving instructions
- Code expansion offset by compacting the useful code, increasing ILP

Results of early Superblock formation

- Forming Superblocks early in the Tree-SSA passes simplifies control flow to provide optimizers longer straight-line code sections

- Case in point: 256.bzip2
  - Improves by 3% with early Superblock formation

256.bzip2

Excerpt from `generateMTFValues`

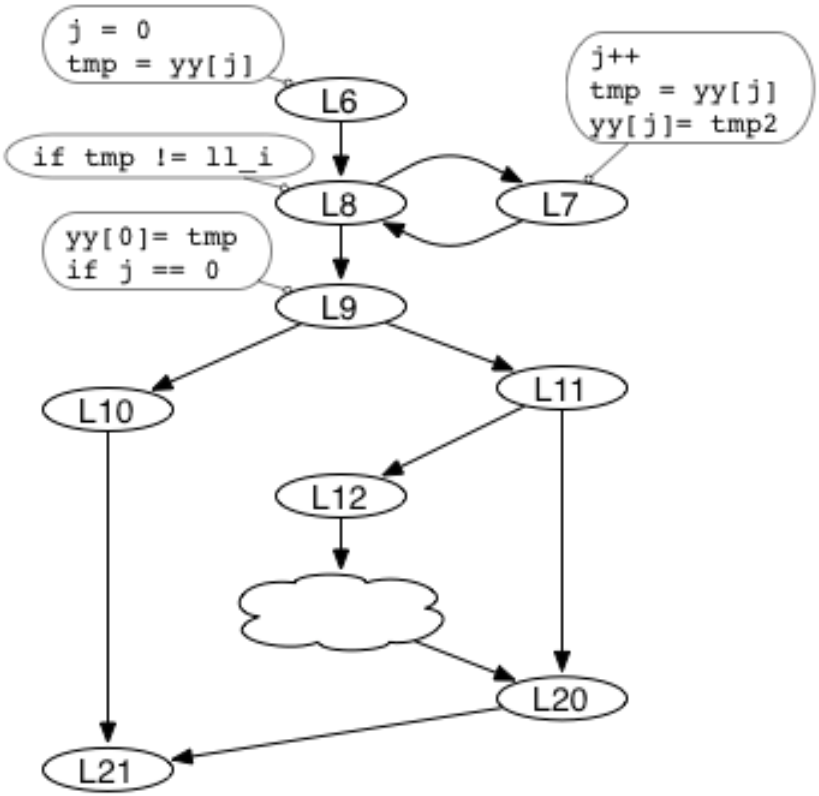
```
L6:   j = 0;
      tmp = yy[j];
L8:   while ( ll_i != tmp ) {
L7:      j++;
         tmp2 = tmp;
         tmp = yy[j];
         yy[j] = tmp2;
      }
L9:   yy[0] = tmp;
      if (j == 0)
L10:     /* do something */
         /* yy is not touched */
      else
L11:     /* do something else */
         /* yy is not touched */
L21:  ...
```

Closely scheduled stores may stall on Itanium 2

Executes only if loop iterates

256.bzip2

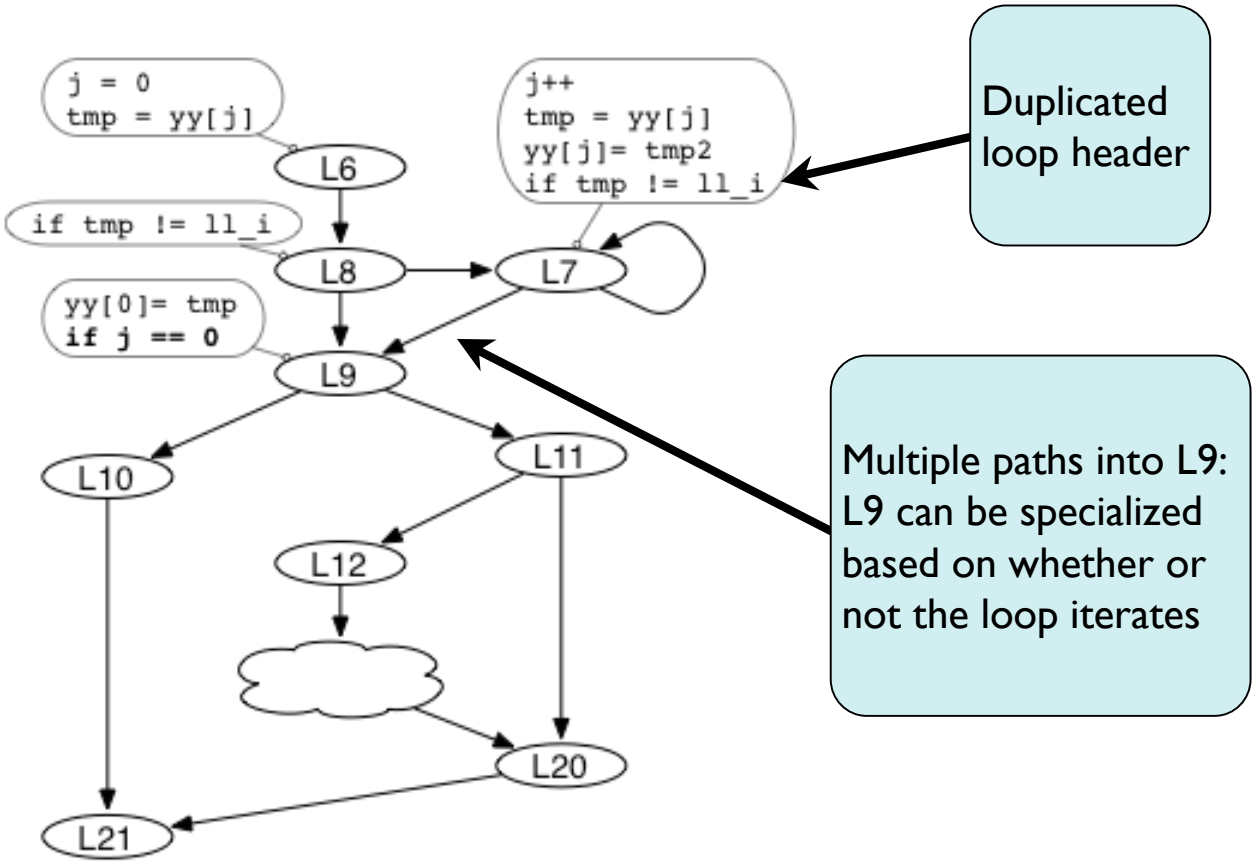
- Immediately after constructing SSA form



IMPACT
ADVANCED COMPILER TECHNOLOGY

256.bzip2

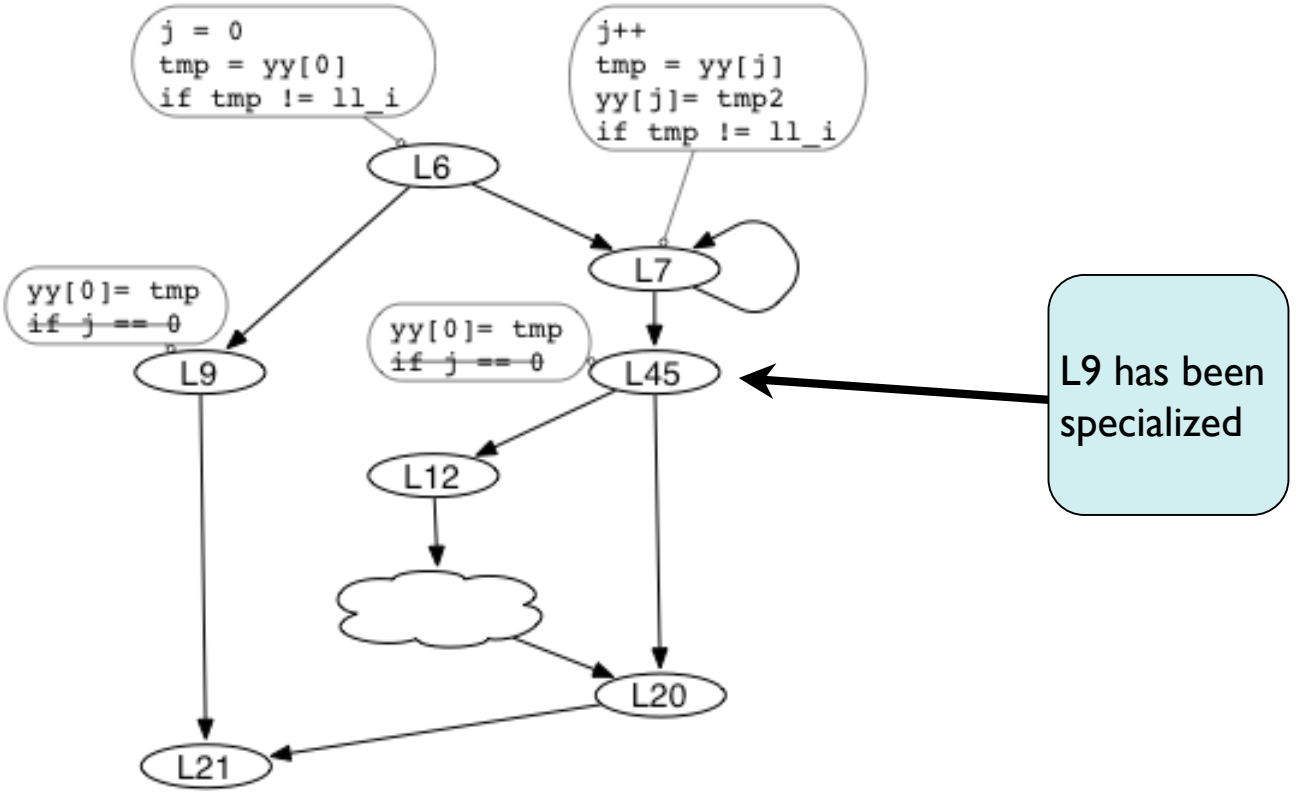
After Superblock formation



IMPACT ADVANCED COMPILER TECHNOLOGY

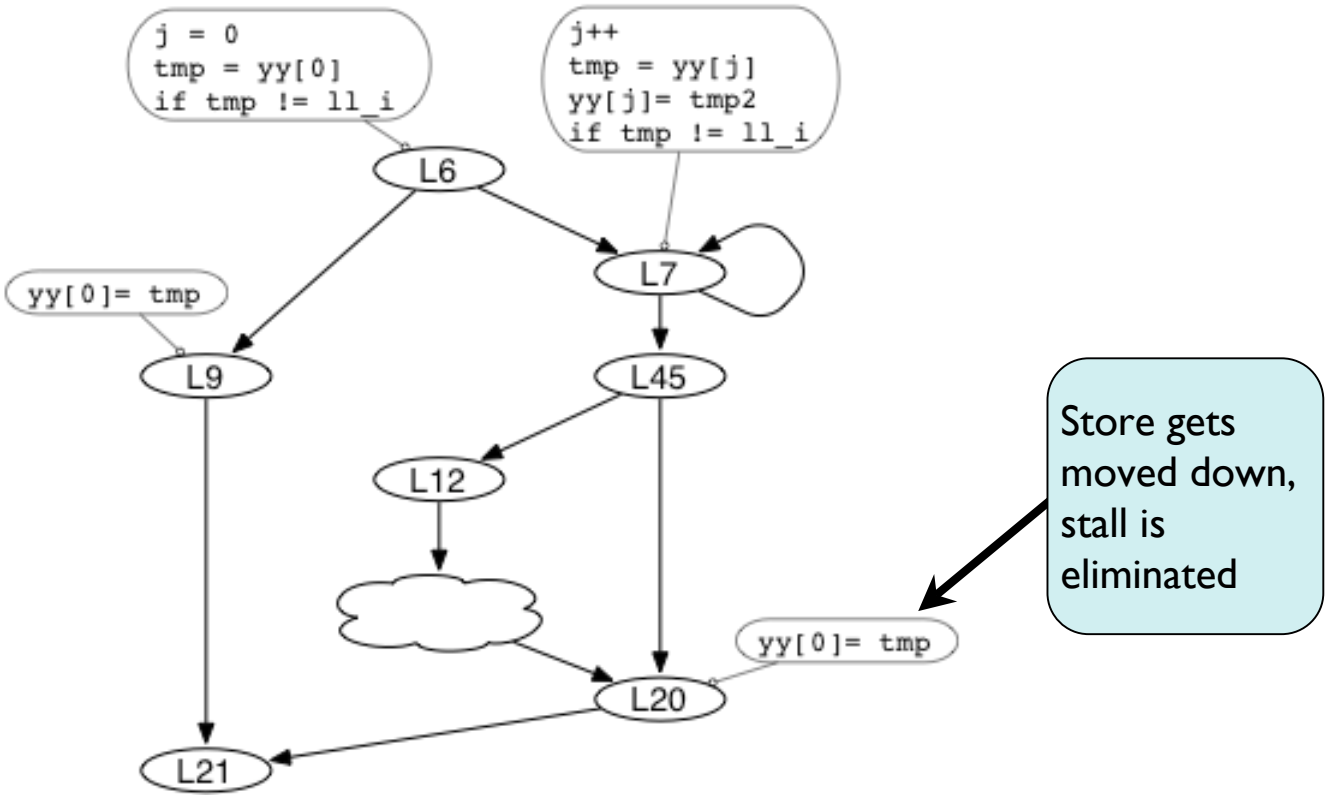
256.bzip2

- After value range propagation

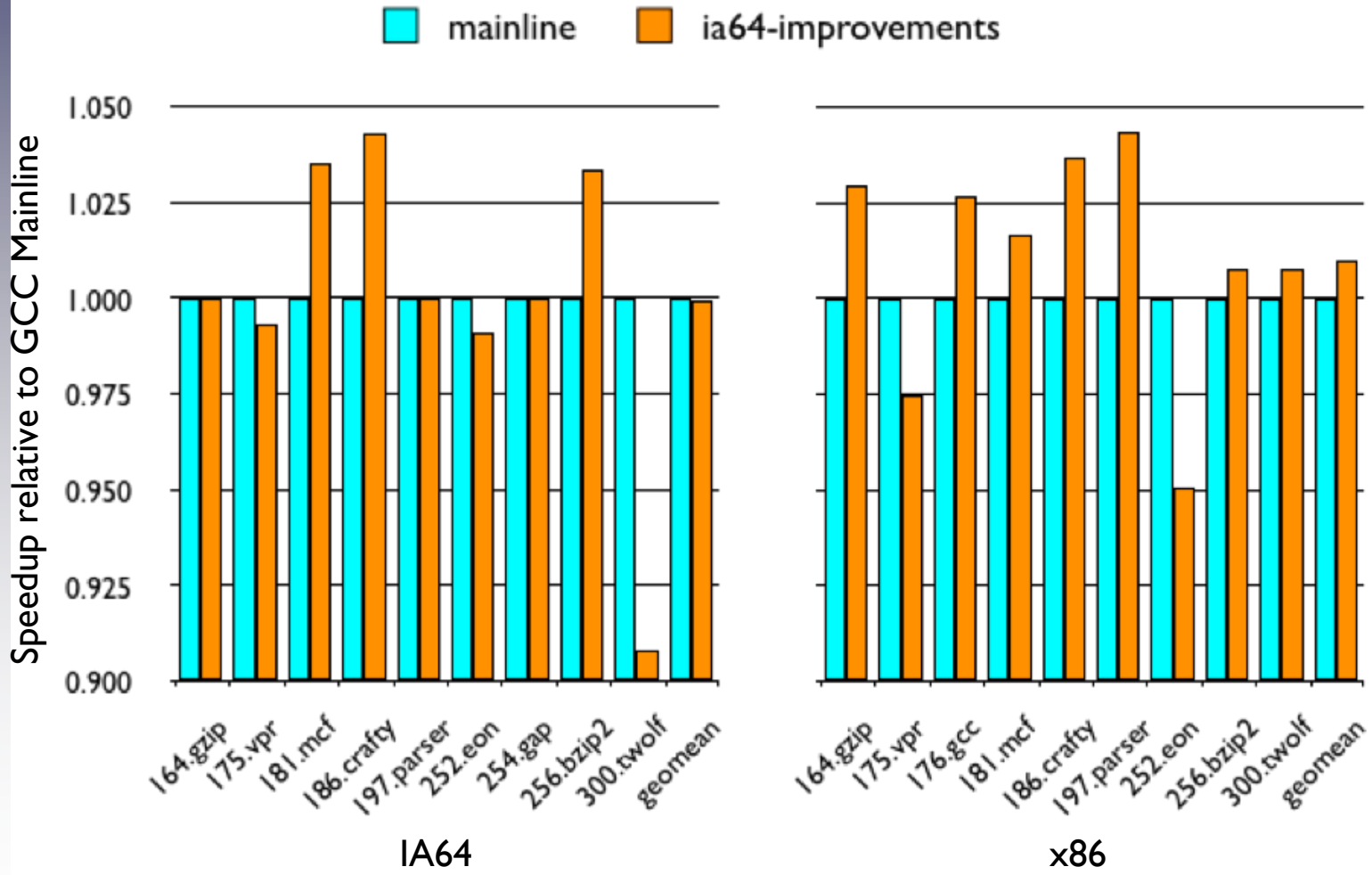


256.bzip2

After store sinking



Results -- Integer

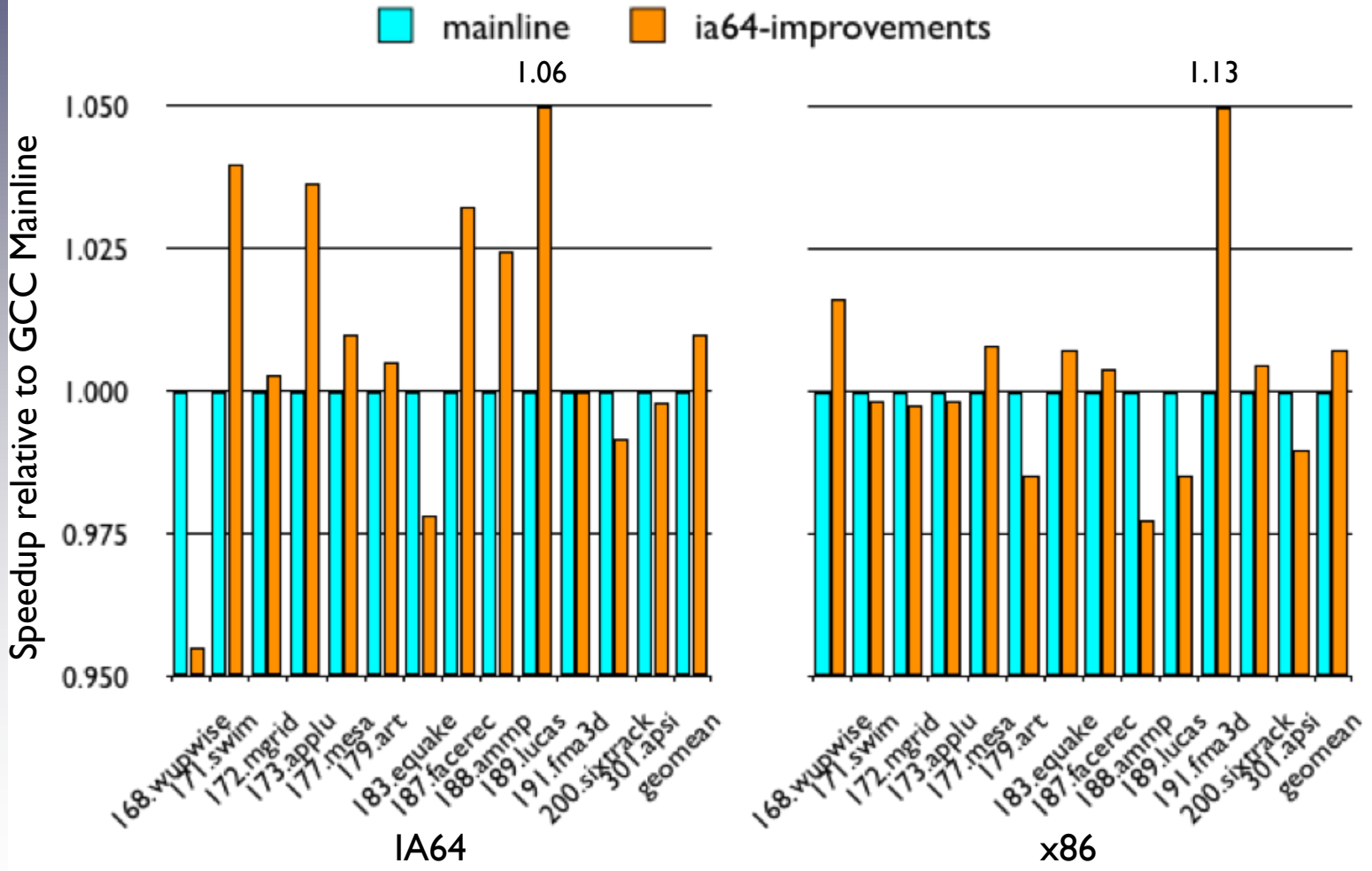


IA64 x86

Estimated SPECint2000 speedup over GCC mainline

IMPACT
ADVANCED COMPILER TECHNOLOGY

Results -- Floating Point



Estimated SPECfp2000 speedup over GCC mainline

IMPACT
ADVANCED COMPILER TECHNOLOGY

Analysis

- 300.twolf on IA64
  - Performance degradation due to a bug in tail duplication
  - Without Superblock formation, partial redundancy elimination is able to register promote a load/store pair inside a hot loop
  - The loop in question is transformed into a non-simple form by tail duplication
- 191.fma3d on x86
  - `platq_stress_integration` sets up symbolic variables using `min`/`max` macros
  - Superblock formation allows constant propagation to simplify the function

186.crafty on ia64

- 186.crafty improves by 4.3% when Superblocks are formed early

	Mainline	ia64-improvements
Useful instructions per cycle	1.161	1.176
Total stalls (fraction of total)	0.495	0.467
Branch mispredictions	19.3e9	18.3e9
L1I hit rate	0.808	0.809
L2I hit rate	0.997	0.998

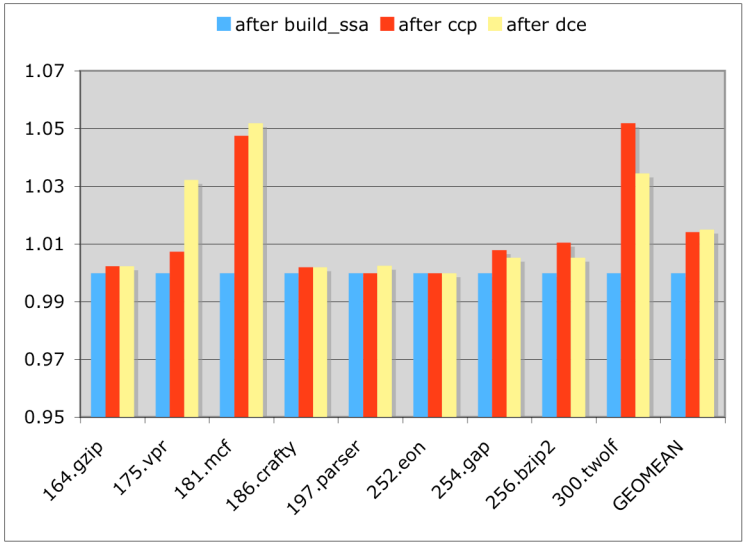
186.crafty on x86

- 186.crafty improves by 3.7%
  - Instruction cache miss rate drops from 7.4 to 6.8%
  - Number of I-cache references drops by one third
  - Binary size decreases by a few KB

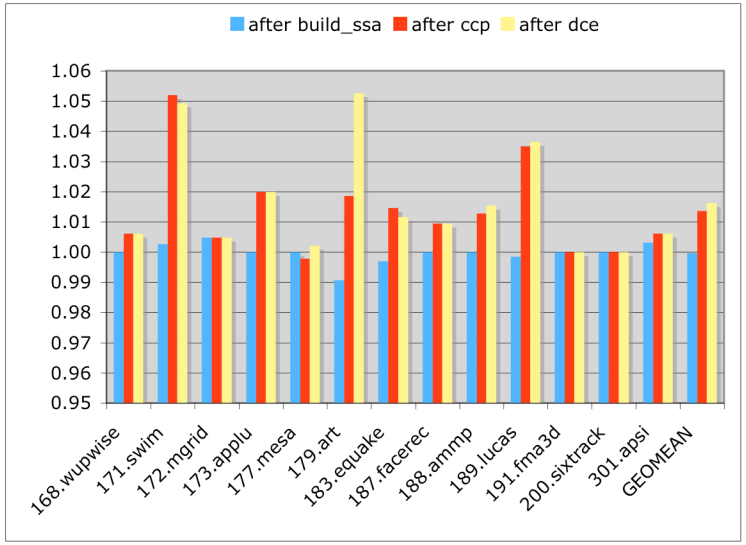
New work

- Experimented with placement of Superblock formation
- Initially placed early -- immediately after building SSA
- Experiment moved Superblock formation to after conditional constant propagation and dead code elimination

Superblock formation without profile info



Integer

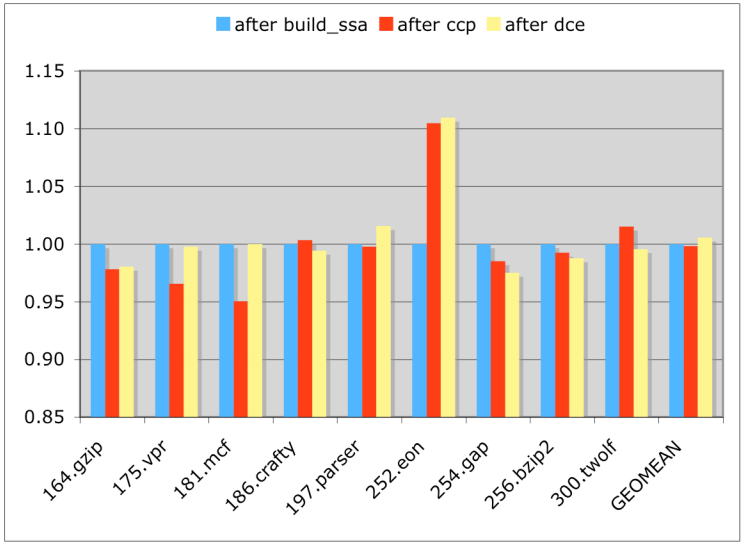


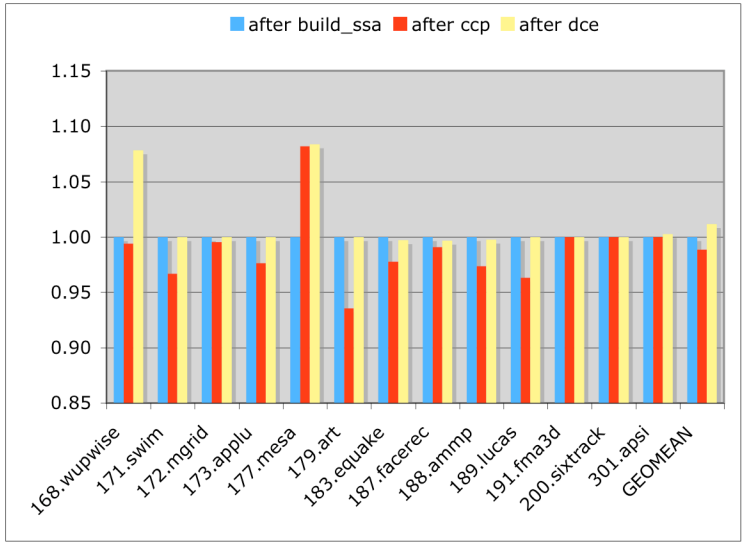
Floating Point

Results are for Itanium

IMPACT
ADVANCED COMPILER TECHNOLOGY

Superblock formation with profile info





Integer

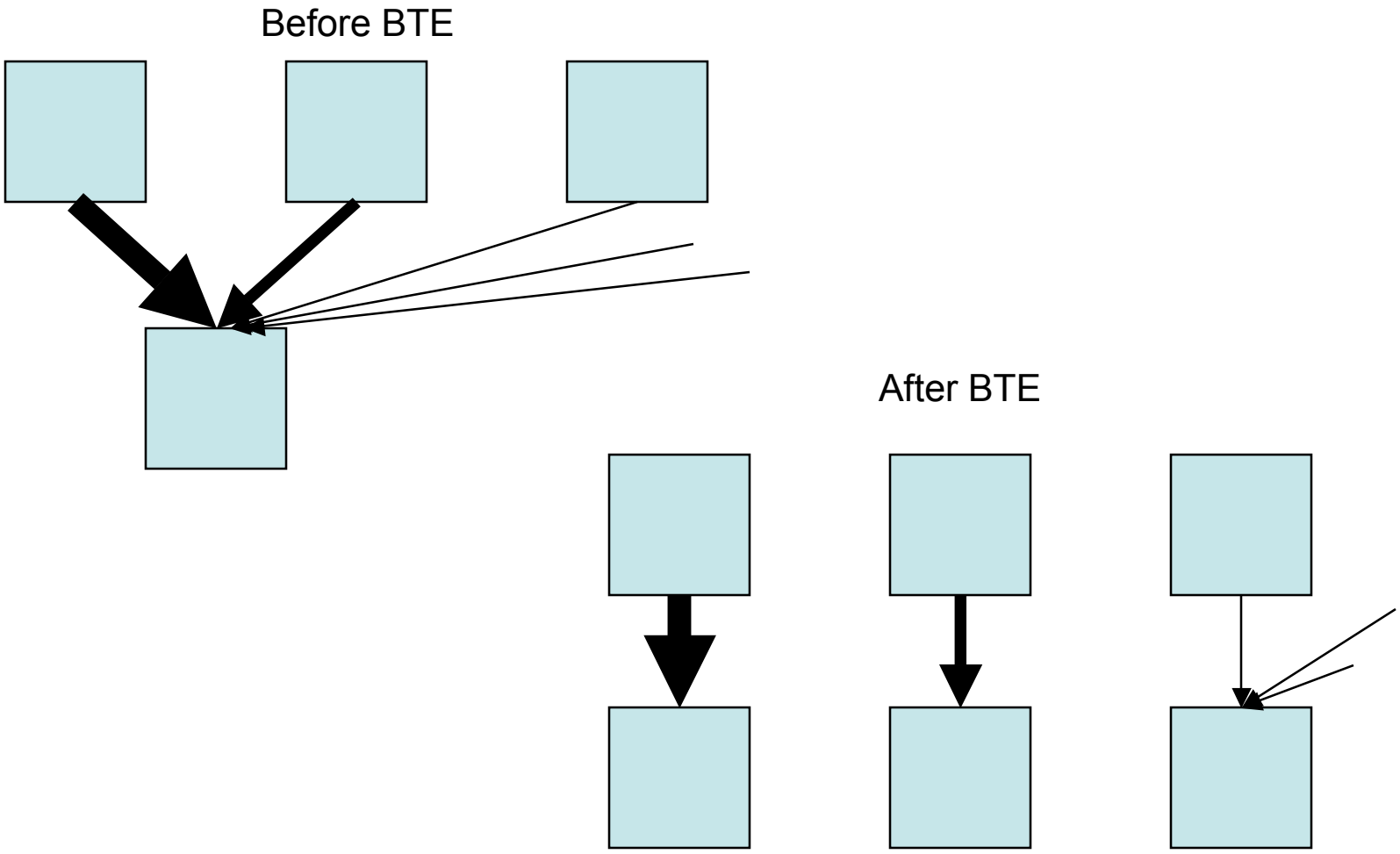
Results are for Itanium

IMPACT
ADVANCED COMPILER TECHNOLOGY

Future work

- Branch target expansion pass
- Finds branches from the tail of one Superblock to another
- Duplicates the target Superblock to form a longer Superblock

Branch Target Expansion



Branch Target Expansion

- Operates on Superblocks instead of basic blocks
- Superblock Tail Duplication
  - A basic block can be tail duplicated at most one time
- Branch Target Expansion
  - A Superblock can be duplicated multiple times
- Much more code expansion
  - More opportunity for optimization
  - Must balance with increased i-cache pressure

Loop Peeling and Unrolling

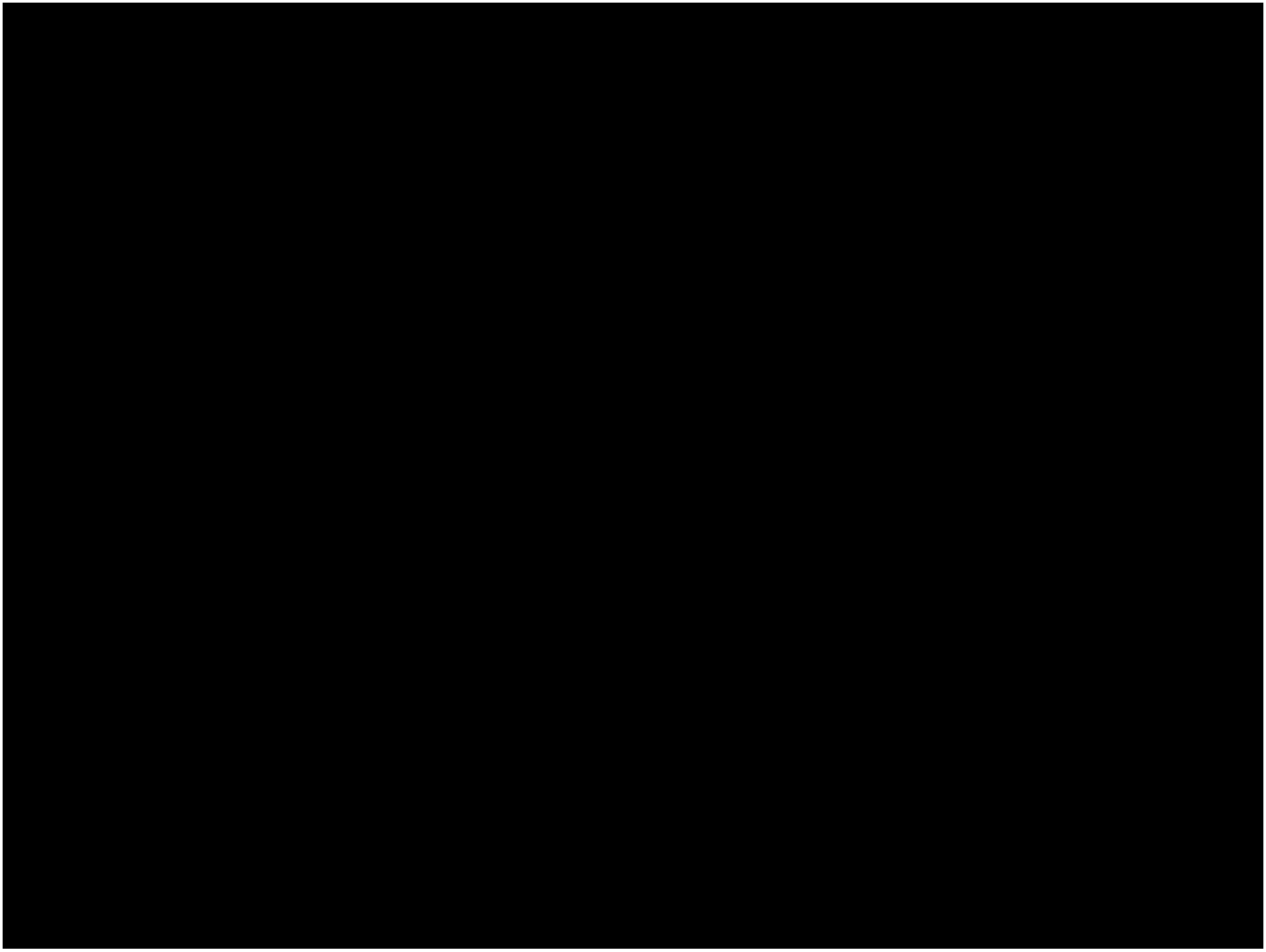
- Loops form barriers to Superblock formation
- Branch target expansion will naturally unroll loops if left unchecked
- For maximum benefit, loop peeling and unrolling should work closely with branch target expansion

Branch Target Expansion pass

- Runs immediately following Superblock formation at the Tree-SSA level
- Subsumes the loop header duplication pass previously implemented in Superblock formation

Status

- Patch has been submitted to gcc-patches
- Testing a small change in response to reviewer feedback
- Will hopefully be approved for 4.3
- Questions?



Future work

- BTE pass is not entirely correct
  - Multiple passes may cause incorrect program execution
- Need to represent the Superblocks to the BTE pass
  - It can be difficult to find the extent of a Superblock given only a member basic block

Future work

- Finish BTE pass
- Investigate combination of loop unrolling and BTE

Loop Peeling

- Looking at moving the loop unrolling and peeling pass forward in the Tree passes
- Naïve attempt resulted in a memory allocation related segfault in a later module